Classification Rule Learning Using Subgroup Discovery of Cross-Domain Attributes Responsible for Design-Silicon Mismatch

Nicholas Callegari[1], Dragoljub (Gagi) Drmanac [1], Li-C. Wang[1], Magdy S. Abadir[2]
[1]University of California - Santa Barbara
[2]Freescale Semiconductors, Inc.

ABSTRACT

Due to the magnitude and complexity of design and manufacturing processes, it is unrealistic to expect that models and simulations can predict all aspects of silicon behavior accurately. When unexpected behavior is observed in the post-silicon stage, one desires to identify the causes and consequently identify the fixes. This paper studies one formulation of the design-silicon mismatch problem. To analyze unexpected behavior, silicon behavior is partitioned into two classes, one class containing instances of unexpected behavior and the other with rest of the population. Classification rule learning is applied to extract rules to explain why certain class of behavior occurs. We present a rule learning algorithm that analyzes test measurement data in terms of design features to generate rules, and conduct controlled experiments to demonstrate the effectiveness of the proposed approach. Results show that the proposed learning approach can effectively uncover rules responsible for the design-silicon mismatch even when significant noises are associated with both the measurement data and the class partitioning results for capturing the unexpected behavior.

Categories and Subject Descriptors

B.8.2 [**Hardware**]: Performance Analysis and Design Aids

General Terms

Algorithms, Performance, Reliability

Keywords

Delay Test, Learning, Timing Analysis, Data Mining

1. INTRODUCTION

As manufacturing technologies advance to smaller size, it becomes difficult to accurately model and predict silicon behavior. Consequently, it is not uncommon that with today's high performance designs, some aspects of silicon behavior are out of expectation in the first-silicon stage. When unexpected silicon behavior is observed on a collection of sample chips, one desires to diagnose to find the causes. The potential causes can be due to unforeseen design issues during design phase or unexpected design-manufacturing interactions. In both cases, the potential search space for causes can be enormous. Furthermore, such a cause may be due to combined factors from multiple steps of design and/or manufacturing processes.

To remove or minimize unexpected behavior, causes are translated into fixes. Fixes could be applied to models or to design. Fixes can be divided into *domains* such as timing, power, layout, mask, etc. In general, a fix can be cross-domain. For example, a cell by itself is not an issue. Only when this cell's output interconnect is implemented with a certain shape of layout causes an issue. A fix can be in the form of a rule. In the simple example, the rule may look like "if cell A followed by layout pattern B, then unexpected behavior occurs with 95% chance". This rule can be used to search a design for places to fix. Once those places are found, the actual action for fixing can be either by replacing the cell or by re-implementing the layout. Note that the example rule is cross-domain, involving *attributes* from timing and layout domains.

Suppose we are given some silicon samples with unexpected behavior presented as measured delays on a set of paths. Suppose the unexpected behavior is that some paths are more under-estimated than others, i.e. predicted delay is less than than measured delay. In this scenario, we desire to discover why those paths are more under-estimated. Our goal is to identify fixes applicable to either the timing model or the design to minimize the under-estimation.

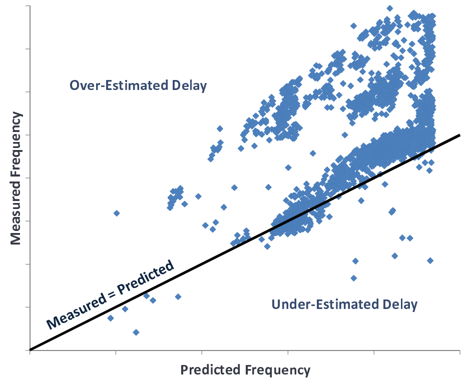

Figure 1: STA vs. Silicon Path Frequency Correlation

To given an example of the unexpected behavior, Figure 1 shows a scatter plot where predicted frequencies are plotted against measured frequencies on 2000 critical paths from a high performance processor design. The design was manufactured with a 65nm technology node. Each measured frequency value is an average taken on 40 sample chips. Predicted frequencies were outputs from a timing analysis tool (STA) run at nominal corner. Because STA was run at nominal corner, one expects that the delays of some paths could be under-estimated. As shown in the Figure, many path delays are under-estimated. However, notice that some paths are much more under-estimated than others. For example, in the Figure a straight line is shown where the measured frequency is equal to the predicted frequency. All paths below the line could be considered as outliers where their delays are more under-estimated. With such a line, the paths are partitioned into two classes, one class containing the outliers and the other with the rest of the population. Then, we ask the question: what are the causes which make those outliers more under-estimated than the rest of the population?

The design-silicon mismatch example describes a scenario where certain silicon measured instances demonstrate behavior that is out of design's prediction (and hence out of one's expectation). The problem is formulated by partitioning all measured instances into two classes where one class contains instances showing the unexpected behavior and the other class has the rest of the population. We call the first *special class* and the second *population class*.

Various works have been proposed to analyze design-silicon mismatch by taking the classification view. For example, the work in [1] was proposed to solve the problem where the special class contains only one instance. The method could produce *ranked hypotheses* to explain why the instance is special. When the special class contains many instances where the number is proportional to the size of the population class, the authors in [2] proposed to analyze the mismatch by ranking design features. The work in [3] solves a similar problem as that in [1]. However, instead of trying to explain the special instances, the method in [3] was proposed only for locating additional instances in a design, similar to the special instances. Hence, it is more for prediction than explanation.

This work is different from prior works in the following aspects. First, our approach finds an explanation for the special class in terms of *rules*. While a *hypothesis* in [1] was defined as a combination of design features, a rule in this work is explicitly represented as a combination of features with additional constraints on the ranges of their values. Second, our approach is designed to handle scenarios where the special class contains multiple instances and the fixes for those instances could also be multiple. In other words, our approach is capable of uncovering a fix shared by multiple instances as well as multiple fixes from groups of instances all contained in a single special class. In contrast, the works in [3] and [2] both did not provide results with such high diagnosis resolution.

In this work, we propose a classification rule learning approach in combination with subgroup discovery, to find rules describing subsets of features belonging to the class. This method significantly differs from those previously introduced for analyzing design-silicon mismatch because of its ability to provide much higher diagnosis resolution. As it will be demonstrated with the experimental result, this higher resolution not only allows accurate identification of relevant design features to the special class but also enables finding exact combinations, or subgroups, of feature value ranges responsible for the design-silicon mismatch.

The rest of the paper is organized as the following. Section 2 gives a short description on the background of learning techniques and related works. Section 3 presents the overall methodology and the extraction of design features. Section 4 presents the proposed rule learning based diagnosis algorithm. Section 5 describes the controlled experiments and explains the results. Section 6 concludes.

2. BACKGROUND AND RELATED WORK

Recent work in this area has been applied to diagnose design-silicon timing mismatch using statistical learning approaches. The authors of [2, 7] use kernel-based learning to rank the importance of design features, such that a feature contributing the most to design-silicon timing mismatch is ranked the highest. Support Vector Machine (SVM), in particular $epsilon$-insensitive regression, was used in addition to multiple kernel variations for correct feature interpretation. The result however, only provides an ordered list of features based on importance. The authors propose if highly ranked features are specific cells, then they could be replaced to fix the problem.

The authors of [1, 3] use kernel-based learning and rule mining for speedpath diagnosis. In the diagnosis application there exists an extremely unbalanced dataset consisting of two classes, speedpaths and non-speedpaths. The number of speedpaths range in 10's to 100's, where the number of non-speedpaths can range in the millions. Given the unbalanced dataset, the authors in [1] identify discrete ranges of features which maximize the separability between the two classes. For example, a speedpath may contain a continuous feature that is the capacitive load for a specific cell. That cell contains significant delay due to mis-modeling in the extreme region above 90% of the total output load capacitance on the cell. The methodology [1] Hypothesis Pruning and Ranking (HPR) is only able to identify this region given the following constraints. (1) The mis-modeling is significant enough that all paths tested would be speedpaths if it contains the feature in this region, i.e systematic mis-modeling effects must always be greater than random noise. (2) The separability of this region to the nearest non-speedpath value is greater than the separability of any other feature.

3. THE METHODOLOGY, FEATURES AND HYPOTHESIS SPACE

Our overall methodology for extracting classification rules is shown in Figure 2. We first begin with the mismatch data consisting of predicted result and measured result on a set of sample chips. As discussed before, instances contained in this data are partitioned into two class, the special class and the population class. Then, a set of design features are used to characterize the potential search space for the fixes of the special class. The concept lattice (as explained later) is used to filter the search space in order to obtain cross-domain features. Classification rule learning is applied to uncover rules based on systematic trends and patterns of interest. The methodology is applicable to all rules and not limited to cross-domain, however given the assumption that cross-domain rules are of most interest for their potential for inaccuracy, we focus the search accordingly.

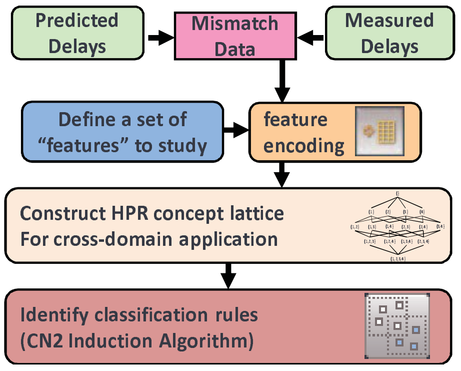
Predicted Delays
Mismatch Data
Measured Delays
Define a set of "features" to study
feature encoding
Construct HPR concept lattice For cross-domain application
Identify classification rules (CN2 Induction Algorithm)

Figure 2: Problem Formulation and Methodology

In this work we apply the methodology to identify timing mismatch for critical paths. The mismatch data is obtained by taking the difference between the predicted and measured delays. We define a set of n descriptive features $f_1, ..., f_n$ which describes the paths. These features are potential sources of uncertainty for all domains. The mismatch data then needs to be converted into a classification problem by partitioning the dataset in a manner such that those paths which contain significant mismatch are classified as special, and the remainder classified as the population. Figure 3 gives an example of partitioning a dataset into two classes of paths. For example, in the Figure a cut is selected at 0ps, and all paths with a positive delay difference are converted into class +1, and those with a negative delay difference are converted into class -1. If our interest is those paths which contain positive mismatch we would label class +1 as the special class, and class -1 as the population class.

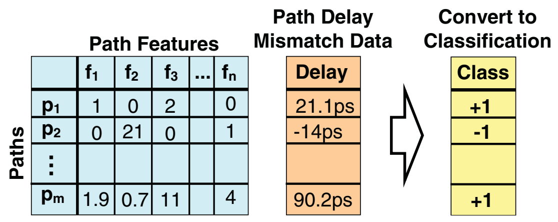

Figure 3: Converting Mismatch Dataset into a Classification Problem

We subsequently separate the features which describe the paths into their appropriate domains. We begin by constructing a concept lattice which is a data structure that consists of all possible combinations of features. In order to focus our search on cross-domain features, we prune the lattice so that only cross-domain features are considered. The pruned concept lattice is shown in Figure 4. In this Figure each value in the set represents the index f_i for all features. The combination of features f_1 and f_2 which would be represented as $\{1, 2\}$ does not exist in the pruned lattice because that combination is within a single domain. However, the combination $\{1, 3\}$ exists in the pruned lattice as these features are cross-domain, consisting of features from both Domain A and Domain B.

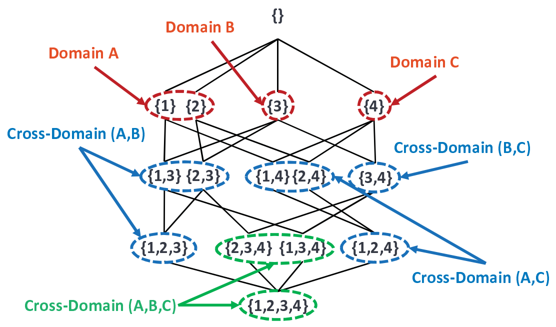

Figure 4: Concept Lattice

From these cross-domain feature combinations, our goal is subgroup discovery to identify which are most important systematic effects. In addition these features are descriptive and therefore continuous, so we want to identify not only the combinations but also the range or subset of each individual feature. For example, there exists significant timing mismatch in a path which is in a specific region of the design, and is connected to a specific power grid. If we have two features, f_1 and f_2 which are the x, y coordinates (layout domain), and a feature f_3, which is the power grid (power domain). Our goal is to not only identify that the combination of these 3 sets of features cause mismatch, but also the specific range, i.e. $\{X\_coord, [3255-4682]\}, \{Y\_coord, [1925-9381]\}, \{P\_Grid, 15\}$.

3.1 Design Features

Introduced by the authors of [2], features are defined as any source of uncertainty in a design. They can consist of two types, *occurrence*-based or *descriptive*-based. Features whose values are boolean are considered occurrence-based, while features whose values are continuous are descriptive-based. An example of an occurrence-based feature can be a specific cell. If that cell appears in a path, the value for the feature is 1, otherwise 0. Adversely a descriptive-based feature may be defined as the capacitive load associated with a specific cell, or X,Y location of a path on the die. In classification rule learning we refer to these types of features as discrete or continuous. We are not constrained by either type of feature however as we explain in section 4.4, special attention is needed in order to extract rules from continuous features.

Although not limited to, we present a collection of potential features to extract from the design process. These features are sorted into 3 domains for which they are associated to.

1. Timing features
   - **Cell-Based**: Type of cells which occur in a path and/or associated drive strength.
   - **Net-Based**: Nets can be grouped based on metal layer, layout shape, length or width. Features can also be defined by RC extraction results.
   - **Transistor-Based**: A path can be viewed from a transistor netlist and the types and/or number of active transistors which occur.
   - **Clock Timing**: The clock tree which launch and capture flip-flops utilize.
2. Layout features
   - **Via-Based**: Type of vias, (i.e. single, double) and the quantity which in a path.
   - **Location Based**: X,Y location of a path (or a cell) on a die (or on a wafer), distance of a path, area of the path on the die.
3. Electrical features
   - **Power Grid Based**: The distance a cell is to a power grid source and associated power rail.
   - **Switching Activity**: The current draw on the power grid given switching activity.

Keep in mind feature definitions are entirely up to the user and can depend on the application. Since features can be closely related to the concerns in modeling, design methodology, and/or process, they are treated as highly-sensitive proprietary information.

4. THE DIAGNOSIS ALGORITHM

Typically to solve this problem, there exist two types of learning approaches, supervised and unsupervised [4]. Classification rule learning, a form of supervised learning, uses predictive induction in order to identify a set of rules, also known as a model, for classification or prediction. Associative rule learning, a form unsupervised learning, uses descriptive induction in order to identify individual rules which define interesting patterns in data.

In this work we adapt and modify the algorithm CN2 proposed by [4] which applies supervised learning from classification rule learning but is adapted to subgroup discovery from associative rule learning in order to achieve both descriptive and predictive induction. The result is the ability to identify individual rules which define interesting patterns for a specific class of data.

Given a set of paths described by features, and a $Class$ for each path, based from mismatch, we are trying to discover a combination of features which are statistically 'most interesting' with respect to the class of interest. This combination of features (attribute-value pairs) is called the condition $Cond$. The discovery of a rule takes the form $Class \leftarrow Cond$, which represents $Cond$ implies $Class$. As rules are induced from a dataset, the process of subgroup discovery is targeted at uncovering the condition of a selected target set of paths with a given class of interest. From this definition subgroup discovery is essentially a form of supervised learning.

There exist two issues with this form of supervised learning. (1) The standard assumption made by supervised learning is to identify rules which produce the greatest accuracy. This is also referred to as overfitting. In order for our method to be effective, it must relax this constraint to filter noise. (2) Typically supervised learning techniques essentially look to cover the targeted paths with a given

$Class$, eliminating them from the search space once covered. However, multiple conditions may exist that cause mismatch which overlap. We account for this by applying a weighted filter instead of a binary covering algorithm, explained in detail in Section 4.3. These two key differences allow our method to still obtain the significance of class, but formulate the rule discovery in a descriptive induction manner, typical of unsupervised learning. The result of using this method, is instead of generating rules to create models, we are looking for rules for patterns of interest, which must be relatively simple.

4.1 Classification Rule Learning

To induce a set of rules, two main procedures are conduced. (1) A bottom-level search and (2) A top-level control procedure repeatedly executes the bottom-level search to generate a set of rules.

The bottom-level search procedure consists of a beam search algorithm which is a optimization of best-first search. Beam search is a graph search that uses a heuristic to determine which edge to traverse. The heuristic used in this method is the accuracy of a propositional classification rule of the form $Class \leftarrow Cond$ is equal to the conditional probability of class $Class$, given the condition $Cond$ is satisfied:

$$Acc(Class \leftarrow Cond) = p(Class|Cond) = \frac{p(Class.Cond)}{p(Cond)} \tag{1}$$

Additionally to test significance of the rule, we use the likelihood ratio statistic (LRS) [6] that measures the difference between the class probability distribution in the set of examples covered by the rule and the class probability distribution in the set of all examples.

$$Sig(R_i) = Sig(Class|Cond_i) = 2 \cdot \sum_j n(Class_j.Cond_i) \cdot log\frac{n(Class_j.Cond_i)}{n(Class_j) \cdot p(Cond_i)} \tag{2}$$

Where for each $Class_j$, $n(Class_j.Cond_i)$ denotes the number of instances in the set where the rule body holds true, $n(Class_j)$ is the number of $Class_j$ instances, and $p(Cond_i)$, which is the probability of $Cond_i$, plays the role of the normalization factor. Note that although for each generated subgroup description one class is selected as the target class, the significance criterion measures the distributional unusualness unbiased to any particular class. As such it measures the significance of the rule only.

The top-level control procedure begins by adding the default rule (providing for the majority class assignment) as the final rule in the induced rule set. The control procedure is iterated inducing rules for each class in turn. In order to prevent the algorithm from finding the same rule again, for each induced rule, covered examples belonging to that class are removed.

4.2 Weighted Relative Accuracy Heuristic

The accuracy of the propositional classification rule proposed in Equation 1, uses the conditional probability in order to calculate rule accuracy. However this accuracy metric contains two fatal flaws. (1) It does not account for the *generality* of the rule which affects the relative size of a subgroup ($p(Cond)$). (2) It does not account for distributional unusualness or *relative accuracy*. We employ weighted relative accuracy in order to address these flaws. Weighted relative accuracy [4] is a variant of rule accuracy that can be applied to subgroup discovery. Weighted relative accuracy is defined as:

$$WRAcc(Class \leftarrow Cond) = p(Cond) \cdot (p(Class|Cond) - p(Class)) \tag{3}$$

Weighted relative accuracy consists of two components, rule *generality*, and distributional unusualness or *relative accuracy*. To address the previous concern (1) the generality component ($p(Cond)$) allows the accuracy of the rule to be deemed interesting only if it improves upon this 'default' accuracy. To address the previous concern (2) the relative accuracy component ($p(Class|Cond) - p(Class)$) measures the difference between true positives and the expected true positives (expected under the assumption of independence of the $Class$ and $Cond$ of the rule). However it is easy to achieve high relative accuracy with highly specific rules i.e. overfitting. To avoid this the generality component shows its effectiveness as a 'weight' so the weighted relative accuracy trades off generality of the rule and relative accuracy.

4.3 Weighted Covering Algorithm

Traditional rule learning algorithms use covering algorithms for set construction [6]. A deficiency of the covering algorithm is that only the first few rules may be of interest as subgroup descriptions with sufficient coverage and significance. In subsequent iterations, rules are induced from biased examples of subsets, i.e. subsets including only examples not covered by previous induced rules.

The covering algorithm is modified so positive examples are not removed shown in Equation 4. Instead the algorithm stores with each example, a count indicating how often (with how many rules) the example has been covered so far. Weights derived from these example counts then appear in the computation of WRAcc. Initial weights are 1, covered can range from 0 to 1, where 1 would represent binary covering, and 0 would represent no covering.

$$WRAcc(Class \leftarrow Cond) = \frac{n'(Cond)}{N'} \cdot \left(\frac{n'(Class|Cond)}{n'(Cond)} - \frac{n'(Class)}{N'}\right) \tag{4}$$

In Equation 4, N' is the sum of all example weights, $n'(Cond)$ is the sum of all covered example weights, and $n'(Class|Cond)$ is the sum of all correctly covered example weights.

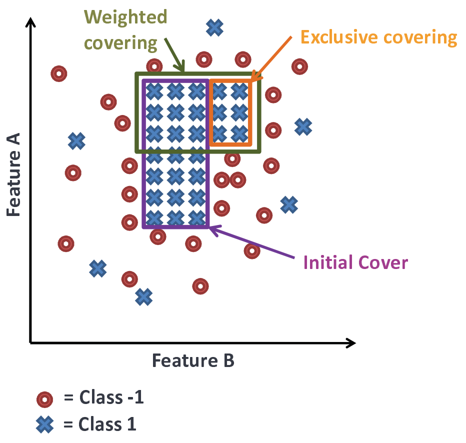

Figure 5: Exclusive vs. Weighted Covering

To best illustrate the benefit of weighted rule covering shown in Figure 5 we represent a two dimensional search space based on Feature A and Feature B, and two classes, paths which contain positive delay mismatch (Class 1) and those which contain negative delay mismatch (Class -1). The goal is to cover as many of the paths which contain positive mismatch, while excluding those that do not. If the initial cover was removed from the search after it was identified with exclusive covering, the remaining exclusive cover would not be complete, and possibly too insignificant to identify. Where as with weighted covering, after the initial cover was reduced in weight, the second weighted cover would be identified.

4.4 Discretizing Continuous Features

In order for classification rule learning to effectively identify discrete ranges in continuous features to provide a high resolution for diagnosis, a discretization method must be used which has knowledge of the dataset. An entropy minimization heuristic developed by [5], is employed for its ability to efficiently discretize a continuous feature into ranges based on similar class. For example the heuristic can easily identify a range within f_1 where all samples belong to a specific class. We can use this methodology and identify ranges for each feature individually, and then apply classification rule learning.

This becomes a difficult task when applied to combinations of features, for example ranges for f_1 and f_2. If the entropy is minimized for each feature individually with no knowledge of any other feature, than the entropy for high order, cross-domain, features is not efficiently minimized. To account for this we perform entropy discretization at each edge in the graph of the classification rule learning. The result is an efficient method which can discretize continuous features for accurate rule extraction for single and high order, cross-domain feature combinations.

5. EXPERIMENTS

To evaluate the effectiveness and accuracy of the methodology, we conducted controlled experiments using an industrial ASIC design. The design consists of 220K gates based on a 90nm technology node. The design netlist, Lef/Def files, and standard cell timing library are used in order to extract features. From the netlist and the Lef/Def files we conducted parasitic extraction to obtain the interconnect RC's (SPEF file) used by an STA tool for path delay prediction. Using the extracted parasitics we ran Static Timing Analysis to obtain the most critical 12,000 paths, where each path was composed of 10 to 35 cells.

In the experiment we assume that design-silicon timing mismatch is due to unmodeled or mismodeled cross-domain systematic and random timing effects. The goal is to identify the most important systematic effects, in the presence of significant noise. To observe the mismatch, we assume that the delays of the critical paths are measured on a set of sample chips, similar to the approach to collect the data shown in Figure 3. Measured path delays are compared to the predicted delays from STA, and their differences are used to observe the timing mismatch.

All cell-based features were extracted from the timing report and timing library. The features included: cell type, cell output transition, and cell $\%Load$ capacitance. The interconnect-based features and location-based features were extracted from the timing report and Def file. These included the number of metal layers and vias that each stage of the path traverses through, the path area, and the X-Y coordinates for each path. All feature values are normalized between 0 and 1 in order to weight the features equally.

The controlled experiment is conducted by randomly associating a systematic timing deviation with a combination of cross-domain features within a specific range of those features' distributions. We refer to this a $rule$. This represents the scenario where, for example, all paths which contain $GateA$ with a capacitive load between $0.35 - 0.47$ and $X - coordinate$ between $0.832 - 0.836$, contains systematic timing deviation. We randomly selected five cross-domain feature combinations, or rules, shown in Table 1. The five rules, Inj_R_1 - Inj_R_5, in the table contain the combination of features, F, and the range of the feature, $[range]$, which contains the mean systematic effect injected. Additionally we have identified the corresponding number of paths the rule appears ,$\#ofSamples$.

In addition we associate significant random timing deviation for each path in order to determine if the systematic rules injected can be discovered in the presence of noise. The random component is generated assuming a normal distribution with a 3σ equal to $\pm20\%$ of the delay for each path. Monte Carlo simulation is run on this new perturbed library to simulate silicon samples. The goal of the methodology is to analyze the mismatch between the original timing library and the simulated silicon samples and identify the systematic rules injected based on their systematic uncertainty in the presence of random noise.

Rule	# of Samples	Systematic Mean Effect
$Inj\_R_1 \Leftarrow F_7[0.39 - 0.98] \wedge F_5[0.48 - 0.75]$	366	37%
$Inj\_R_2 \Leftarrow F_{28}[0.42 - 0.93] \wedge F_{38}[0.14 - 0.74]$	199	32%
$Inj\_R_3 \Leftarrow F_{14}[0.41 - 0.52] \wedge F_9[0.13 - 0.96]$	263	36%
$Inj\_R_4 \Leftarrow F_{10}[0.44 - 0.83] \wedge F_{39}[0.32 - 0.73]$	802	38%
$Inj\_R_5 \Leftarrow F_{33}[0.08 - 0.82] \wedge F_{12}[0.19 - 0.60]$	634	39%

Table 1: Injected Rules with Systematic Timing Effects

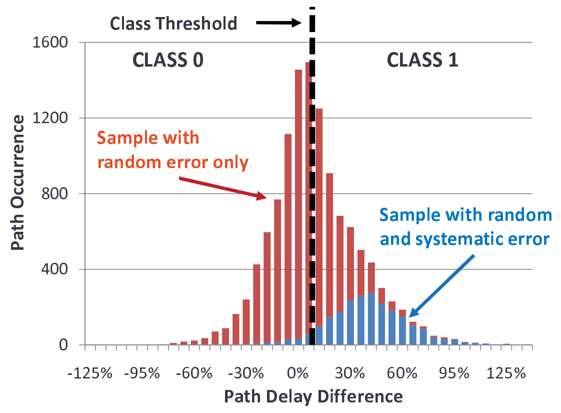

Figure 6: Histogram of Path Delay Mismatch

Figure 6 shows the resulting path delay differences for the 12,000 paths. In this histogram, the x-axis is the path delay difference, and they y-axis is the number of paths. The paths which contain both the random and systematic effects are labeled separate from those which contain only random effects.

# of Samples			
12009			
Class 1		Class 0	
5649		6360	
True Positive	False Positive	True Negative	False Negative
2159	3490	6195	165

Table 2: Data After Random and Systematic Effects

As this is a classification problem, we split the paths into two classes, class 0 and class 1, based on the mean path delay difference of 7%. The goal is to identify rules from class 1 as they are paths most responsible for limiting silicon performance. Ideally class 1 would only contain paths which contain systematic effects, however to best simulate real silicon, many sources are unknown and represented as random effects. Shown in Table 2, we can see the result of the classification based on the random effects. We have included such significant random delay that there exists more samples in class 1 which do not contain systematic delay (false positive), than those that do (true positive). Adversely there exist paths which contain systematic delay misclassified in class 0 (false negative), as well as those which do not (true negative). We separate the data in this way to illustrate the methodology's diagnosis capability in the case where class separation is not easily distinguishable.

5.1 Results

We apply classification rule learning on this dataset and identify the top five predicted rules in no specific order, shown in Table 3. The top five predicted rules are ordered to align with the injected rules from Table 1 for comparison. In analyzing this comparison we found the methodology extremely accurate in identifying the exact combination of features, i.e. the predicted rule contains the same features as the injected. In addition the ranges within the rule for each feature show a strong similarity.

Rule
$Pred\_R_1 \Leftarrow F_7[0.29 - 1.00] \wedge F_5[0.44 - 0.74]$
$Pred\_R_2 \Leftarrow F_{28}[0.41 - 0.59] \wedge F_{38}[0.14 - 1.00]$
$Pred\_R_3 \Leftarrow F_{14}[0.40 - 0.51] \wedge F_9[0.17 - 1.00]$
$Pred\_R_4 \Leftarrow F_{10}[0.01 - 1.00] \wedge F_{39}[0.32 - 0.67]$
$Pred\_R_5 \Leftarrow F_{33}[0.08 - 1.00] \wedge F_{12}[0.18 - 0.60]$

Table 3: Top 5 Diagnosed Systematic Effects

From these results, however, we do find a difference in ranges between the rule Inj_R_2 and $Pred\_R_2$ which lacks the precision of the other four predicted rules. To analyze why our approach could not provide a more accurate rule, we examined the dataset. Shown in Figure 7 we can see the hypothesis space of the two feature combination F_{38} and F_{28} where the injected rule is outlined, as well as the predicted rule result from the analysis. We can see that the algorithm was unable to obtain accuracy for the upper bound of the ranges because of the sparsity of the dataset. However, we notice in the dense region of the lower bounds exhibit great accuracy.

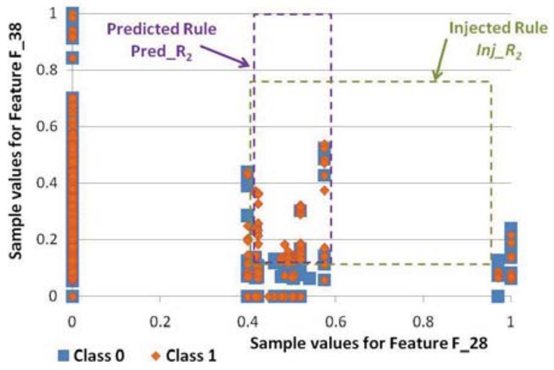

Figure 7: Injected and Diagnosed Rules for $Pred\_R_2$ and Inj_R_2

We examine the distribution of both features F_{38} and F_{28}, shown in Figure 8. In this graph, the x-axis is the sample values, and the y-axis is the feature occurrence. From this graph we can see a very uneven distribution for both sets of features, specifically a significant lack of samples for both classes above the value 0.6.

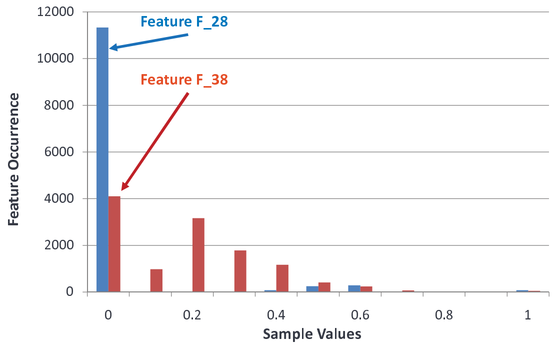

Figure 8: Histogram of Values for Features F_{28} and F_{38}

The sparsity makes it very difficult to derive rules with high accuracy. Additionally the difficulty is even higher when considering that the approach must also identify the upper and lower bounds in a range of data. However, with these extreme complexities our approach was able to identity the exact feature combination for the rule which only existed 199 samples out of a total of 12,000. In comparison to previous works in section 2 this approach provides significant increase in resolution, where the other approaches only rank each individual feature, without identifying feature combinations or a discrete subgroups.

Rule	Class 1 Samples Predict/Inject	Class 0 Samples Predict/Inject	Total Samples Predict/Inject	Total Samples Accuracy
$R1$	326/326	40/40	366/366	100%
$R2$	181/187	10/12	191/199	96.0%
$R3$	236/237	26/26	262/263	99.6%
$R4$	750/755	47/47	797/802	99.4%
$R5$	594/594	40/40	834/834	100%

Table 4: Accuracy of Top 5 Diagnosed Rules

Additionally we examined how many of the samples with the injected rule was also predicted, shown in Table 4. The samples are separated by class based on the threshold from Figure 6. It is important to note that there is a significant number of samples in class 0 due to random noise and the class threshold selected prior to rule learning. The table also combines class 0 and 1 into the total samples predicted and the corresponding percentage. From this table we discover two key findings. (1) The rule learning algorithm is able to identify accurate rules even the class threshold does not precisely separate all samples with systematic error into the same class and (2) The worst case accuracy, rule R_2, provided an amazing accuracy of 96% and more than 99.4% accuracy across the other four rules.

6. CONCLUSION

This paper presented a novel rule learning algorithm that analyzes test measurement data in terms of design features to generate rules. We conducted controlled experiments to demonstrate the effectiveness of the proposed approach. Result showed that the proposed learning approach can effectively uncover rules responsible for the design-silicon mismatch even when significant noises are associated with both the measurement data and the class partitioning results for capturing the unexpected behavior.

7. REFERENCES

[1] N. Callegari, P. Bastani, L. Wang, “Speedpath analysis based on hypothesis pruning and ranking,” *Proc. DAC* 2009.

[2] P. Bastani, N. Callegari, L. Wang, and M. Abadir. "Statistical diagnosis of unmodeled systematic timing effects," *Proc. DAC*, 2008.

[3] P. Bastani, K. Killpack, L.-C. Wang, and E. Chiprout, "Speedpath prediction based on learning from a small set of examples," *Proc. DAC*, 2008.

[4] N. Lavrac, B. Kavsek, P. Flach, and L. Todorovski, "Subgroup discovery with CN2-SD," *Journal of Machine Learning Research*, Vol 5, 2004, pp. 153Ű188.

[5] U.M Fayyad and K.B. Irani "Multi-interval discretization of continuous-valued attributes for classification learning," *IJCAI-93*, 1993.

[6] P. Clark, and T. Nibblet, "The CN2 induction algorithm," *Machine Learning*, Vol 3, (4), 1989, pp. 261-283.

[7] P. Bastani, N. Callegari, L. Wang, M. Abadir, "An Improved Feature Ranking Method for Diagnosis of Systematic Timing Uncertainty," *VLSI-DAT* 2008.

[8] V. Vapnik, "The nature of Statistical Learning Theory," 2nd ed., *Springer*, 1999.

[9] Chengqi Zhang and Shichao Zhang. "Association Rule Mining, Models and Algorithms," *Lecture Notes in Computer Science* Vol. 2307, Springer 2002.

[10] S. Brin, R. Motwani, et al. "Dynamic itemset counting and implication rules for market basket data," *Proc. ACM SIGMOD*, 1997, pp. 255-264.

[11] R. Agrawal, T. Imielinksi, and A. Swami. "Mining association rules between sets of items in large databases," *Proc. ACM SIGMOID*, 1993, pp. 207-216.

[12] D. Burdick, M.Calimlim, and J. E. Gehrke. "MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases," *International Conference on Data Engineering*, 2001, pp. 443-452.

[13] H. Mannila, et al. Discovery of Frequent Episodes in Event Sequences," *Data Mining and Knowledge Discovery*, Vol 1, (3), 1997, pp. 259-289.

[14] M. Vannucci and V. Colla. "Meaningful discretization of continuous features for association rules mining by means of a som," *ESANN*, 2004, pp. 489-494.